

10-A145 839

PROGRAMMING EFFORT ESTIMATION(U) PURDUE RESEARCH  
FOUNDATION LAFAYETTE IN S D CONTE ET AL. 14 AUG 84  
ARO-18691.1-EL DAAG29-82-K-0071

1/1

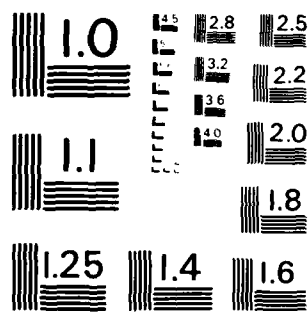
UNCLASSIFIED

F/G 9/2

NL



END  
DATE  
FILMED  
10-84  
DTIC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A145 839

DTIC FILE COPY

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

2

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ARO 18691.1-EL	2. GOVT ACCESSION NO. N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) Programming Effort Estimation		5. TYPE OF REPORT & PERIOD COVERED 15 Feb 82 - 14 Aug 84 Final Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) S. D. Conte, H. E. Dunsmore, V. Y. Shen		8. CONTRACT OR GRANT NUMBER(s) DAAG29-82-K-0071
9. PERFORMING ORGANIZATION NAME AND ADDRESS Purdue Research Foundation Purdue University		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office Post Office Box 12211 Research Triangle Park, NC 27709		12. REPORT DATE
		13. NUMBER OF PAGES 8
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) NA		
18. SUPPLEMENTARY NOTES The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Programming Models Project Management		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) In this research project we proposed to "investigate various models of programming effort estimation and prediction at various stages of the software development process". The project has led to several results and models. This report concentrates on results from the last year of our study (August, 1983 - August, 1984).		

84 9 21 111

DP FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

FINAL REPORT

(TWENTY-FIVE COPIES REQUIRED)

1. ARO PROPOSAL NUMBER: 18691-A-EL
2. PERIOD COVERED BY REPORT: <sup>15 Feb 82 - 14 Aug 84</sup> ~~January 1, 1984 - August 14, 1984~~
3. TITLE: PROGRAMMING EFFORT ESTIMATION
4. CONTRACT OR GRANT NUMBER: DAAG29-82-K-0071
5. NAME OF INSTITUTION: Purdue Research Foundation, Purdue University
6. AUTHOR(S) OF REPORT: S. D. Conte, H. E. Dunsmore, and V. Y. Shen
7. LIST OF MANUSCRIPTS SUBMITTED OR PUBLISHED UNDER ARO SPONSORSHIP DURING THIS PERIOD, INCLUDING JOURNAL REFERENCES:

Conte, S. D., H. E. Dunsmore and V. Y. Shen. Software effort estimation and productivity. To appear in *Advances in Computers* Vol. 24 (M. Yovits, ed.), Academic Press, NY (1984).

Dunsmore, H. E. Software metrics: an overview of an evolving methodology. *Information Processing and Management* Vol. 20, No. 1-2 (1984), 183-192.

Dunsmore, H. E., V. Y. Shen and S. D. Conte. A comparison of a few effort estimation models. *Journal of Parametrics* Vol. 4, No. 1 (1984), 4-14.

Shen, V. Y., T. J. Yu, S. M. Thebaut and L. R. Paulsen. Identifying error-prone software: an empirical study. Submitted to *IEEE Transactions on Software Engineering* (May, 1984).

Thebaut, S. M. and V. Y. Shen. An analytic resource model for large-scale software development. *Information Processing and Management* Vol. 20, No. 1-2 (1984), 295-315.

Volpano, D. M., and H. E. Dunsmore. Empirical Investigation of COBOL Features. *Information Processing and Management* Vol. 20, No. 1-2 (1984), 277-291.

Wang, A. S. and H. E. Dunsmore, Back-to-front programming effort prediction. *Information Processing and Management* Vol. 20, No. 1-2 (1984), 139-149.

Wang, A. S. The estimation of software size and effort: an approach based on the evolution of software metrics. Ph.D. Thesis, Department of Computer Science, Purdue University (August 1984).

Yu, T. J. Software metrics data collection. CSD-TR-421, Department of Computer Science, Purdue University, West Lafayette, IN (November 1982, revised 1984).

8. SCIENTIFIC PERSONNEL SUPPORTED BY THIS PROJECT:

Principal Investigators: S.D. Conte, H.E. Dunsmore, V.Y. Shen

Research Assistants: B.A. Nejme, S.M. Thebaut, A.S. Wang, T.J. Yu

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



## BRIEF OUTLINE OF RESEARCH FINDINGS

### Introduction

In this research project we proposed to investigate various models of programming effort estimation and prediction at various stages of the software development process. The project has led to several results and models. This report concentrates on results from the last year of our study (August, 1983 - August, 1984).

### Large-Scale Software Development Models

Large-scale software development models are models that explain the effort in constructing software products involving a team of programmers. Each such model generally has as parameters the following:

- $S$  - size in thousands of lines of code
- $E$  - development effort in programmer-months
- $D$  - development duration in months.

We have investigated a number of large scale software development models. In this document we report on three of the best known: the Doty model [Herd 77], Putnam's software equation [Putnam 78] (the basis for the SLIM model), and Boehm's COCOMO model [Boehm 81].

The Doty model has a simple form for large-size projects:

$$E = 5.288 \times S^{1.047}$$

Putnam's software equation can be written as

$$E = k \times \frac{S^3}{D^4}$$

where  $k$  is a "technology" constant. Boehm's COCOMO (COntstructive COst MOdel) can have many input parameters in its "Intermediate" form

$$E = a_i S^{b_i} m(X)$$

in which  $a_i$  and  $b_i$  change with three development modes and  $m(X)$  is the product of fifteen cost driver attributes.

During the course of this research we have developed at Purdue University the COPMO (COoperative Programming MOdel) [Thebaut 83, 84]:

$$E = E_p(S) + E_c(\bar{P})$$

in which  $E_p(S)$  is the contribution to total effort from programming a project of size  $S$  and  $E_c(\bar{P})$  is that portion of total effort required by the necessary coordination of the individual efforts of a programming team with average team size  $\bar{P}$ .

In evaluating models we use the metrics.

$\overline{MRE}$  - the mean magnitude of relative error, i.e., the mean percentage of error between a model's predicted effort and the actual recorded effort,

and

$PRED(.25)$  - the percentage of predicted effort values that fall within 25% of the actual recorded effort values.

Using a database we have gathered of 189 industry, military, and university projects, we examined the performance of each of the models. The results appear in Table 1 below.

Table 1.

	$\overline{MRE}$	$PRED(.25)$
Doty	1.85	21%
Putnam	1.16	6%
COCOMO	.79	38%
COPMO	.41	45%

Note that the performance of the COCOMO model is somewhat better than that of the Doty and Putnam models (both of which are clearly unacceptable). Furthermore, note that COPMO is somewhat better still than COCOMO in terms of both mean magnitude of relative error and percentage of estimates that fall within 25% (an acceptable range) of their actual effort values.

In order to make the COPMO an even better predictor of programming effort, we recognized that a single model may not be acceptable for wide ranges of programming productivity. This led to the Generalized COPMO

$$E = b_i S + c_i \bar{P}^{1.5}$$

in which  $b_i$  and  $c_i$  are allowed to vary depending upon the productivity of the programmers involved. For the 189 projects for which we have data this led us to the ten classes and  $b_i$  and  $c_i$  values shown in Table 2.

Table 2.

<i>LOC /MM</i>	$b_i$	$c_i$
0-85	4.7	3.6
86-200	3.0	2.0
201-300	2.4	1.5
301-400	2.2	1.2
401-500	1.8	1.0
501-600	1.6	0.9
601-700	1.2	0.8
701-800	1.1	0.7
901-1000	0.9	0.5
1001-	0.8	0.4

Note that in Table 2 *LOC /MM* refers to lines of code per programmer month.

This Generalized COPMO leads to a new entry in Table 1 as shown in Table 3.

Table 3.

	$\overline{MRE}$	$PRED (.25)$
Doty	1.85	21%
Putnam	1.16	6%
COCOMO	.73	38%
COPMO	.41	45%
Generalized COPMO	.21	75%

Note that the Generalized COPMO outperforms all other models (including the COPMO from which it was derived). This performance of .21 mean magnitude of relative error and 75% acceptable estimates seems to us a step toward a useful effort estimator. On the other hand, note that the information used to obtain the productivity figures were obtained from the project data. It remains for us to show that the estimates will be as good when  $a_i$  and  $b_i$  are derived from historical data alone.

#### Early Size and Effort Estimation

In our continuing research on software development models research we have seen that most models employ size as one of the most important parameters, i.e.,

$$E = f (SIZE, others).$$

Thus, for most models accurate effort estimation relies on accurate size estimation and errors in size estimation will lead to errors (often significant ones) in effort estimation.

Our work in early size and effort estimation has proceeded from the basic idea that we should be able to observe (and measure) some metric early in the software development process that will lead to acceptable final size and total effort estimates. Furthermore, if this process is repeated at various stages during the software development process, we should be able to refine these estimates toward greater accuracy as software development proceeds.

In our most recent (and most successful) in this area we employed the following ideas and hypotheses [Wang 84]:

- (1) In our experimental work our subjects used an "incremental" strategy of producing the program routine-by-routine (along with data structures for the next lower level as discussed below). In this strategy, each subprogram was tested before proceeding to the next. We hypothesized (and found it to be supported by our data) that *LOC* (the number of lines of code in the program) evolves linearly under such a strategy. That is, the "curve" showing lines of code in place in the program increases at an approximately constant rate throughout program development.
- (2) We hypothesized that *VARs* (the number of variables used in a program) is a size-related metric. In our experimental work a "top-down data-structure-first" development strategy was used in which subjects introduced the main software routine and the data items for the next level in a recursive manner. During the software development process *VARs* evolved in a concave-down manner (as we had assumed). That is, a curve showing variables in place in the program increases very rapidly from zero and later flattens out as few variables are added in the latter stages of software development.
- (3) These evolutionary metric curves form useful "finger prints" of the development process that can be used by the programmer and manager (in our case experimenter) alike.

In our empirical investigation we employed forty-four Computer Science graduate students at Purdue University in the summer of 1983. Each was to construct two approximately 400-line Pascal programs. Each subject was allowed a two to three week development time (25 to 35 hours programming time) for each program.

Our hypotheses concerning curve forms for *LOC* and *VARs* were supported and we arrived empirically at the following functional forms for modelling the evolution of *LOC* and *VARs*:

$$LOC(t) = a \times t^{1.2}$$

Note that the 1.2 exponent is very close to linear (i.e., 1.0) while reflecting a slight period of inactivity at the beginning of each software development process.

$$VARs(t) = a \times t^{.2}$$

Note that the exponent .2 suggests a very concave-down curve.

Our early size estimation results appear in Table 4.

Table 4.

Program	Estimate	$\overline{MRE}$	PRED (.25)
1	$\hat{LOC}_{VARS}$	.22	68%
	$\hat{LOC}_{PGMR}$	.23	57%
2	$\hat{LOC}_{VARS}$	.24	61%
	$\hat{LOC}_{PGMR}$	.42	52%

$\hat{LOC}_{PGMR}$  is the size estimate based on an early interview of each programmer on each project.  $\hat{LOC}_{VARS}$  is an early size estimate obtained at the same early interview time by observing the evolution and current status of the metric VARS. Note that size estimates based on VARS outperform the subjects' own subjective size estimates for both programs.

Our early effort estimation results appear in Table 5.

Table 5.

Program	Estimate	$\overline{MRE}$	PRED (.25)
1	$\hat{E}_{VARS}$	.24	64%
	$\hat{E}_{PGMR}$	.42	57%
2	$\hat{E}_{VARS}$	.28	52%
	$\hat{E}_{PGMR}$	.86	9%

$\hat{E}_{PGMR}$  is the programmer's own effort estimate obtained in the early interview of each programmer on each project.  $\hat{E}_{VARS}$  is the effort estimate obtained at the same early interview time by observing the evolution and current status of the metric VARS. Note that effort estimates based on VARS outperform (quite a bit) the subjects' own subjective effort estimates for both programs.

We consider that this research has shown the possibility of using our evolution model of the interaction between data structures and size as a tool for early determination of the final size and total effort of the software development process. More studies toward confirmation must be performed.

### Software Defects

In the experiment involving the forty-four subjects at Purdue University in the summer of 1983 we considered that *programming* ended when programs ran correctly on some standard acceptance test cases. During *testing* subjects ran their programs against some additional standard test cases and made necessary alterations. We investigated several parameters including "time spent in programming" and the "exhaustiveness of the acceptance test cases", but we found no relationship other than our simplest hypothesis - the correlation between testing time and defects discovered during testing was a significant .47.

In an industrial study involving the analysis of 1428 program modules written in Pascal, PL/1, and Assembly language, we investigated the factors affecting  $C_{MD}$ , the count of distinct module defects [Shen 84]. We found that  $\eta_2$ , the number of unique operands in a program, and  $DE$ , the total number of decisions (i.e., Boolean expressions) in a program were the best estimators of  $C_{MD}$ . Note that  $\eta_2$  is very strongly related to *VARs* that proved so successful in the size and effort estimation study reported above.

### Supporting Analyzers and Software Metrics Data Collection

In order to conduct our software metrics work we have produced software analyzers - counters that compute basic metrics for programs written in the languages Fortran, Cobol, Pascal, and C. More than fifty of these have been distributed to interested groups in the military, industry, and universities.

Our Software Metrics Data Collection is a large, comprehensive set of data representing nineteen different program development histories from military, industry, and university projects.

### Future Research

- (1) We intend to continue our work on the Generalized COoperative Programming MOdel. We are looking for an early way of determining the complexity classes.
- (2) Our early size and effort estimation work will continue. We need to
  - (a) refine our models,
  - (b) obtain "real world" verification by the use of non-university project data,
  - (c) continue to investigate development strategies for better estimation and control, and
  - (d) refine the use of evolution curves as "finger prints" of the software development process.
- (3) We want to investigate further the area of software defects. Immediately we see a need to
  - (a) refine the definition of "defects",
  - (b) determine the best predictors of defects,
  - (c) determine the relationship(s) among effort, complexity, and defects,
  - (d) determine how best to split time into development and testing phases, and
  - (e) investigate program design languages and techniques that "avoid" defects.

## References

- [Boehm 81] Boehm, B. W. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ (1981).
- [Herd 77] Herd, J. R., Postak, J. N., Russell, W. E., and Stewart, K. R. Software cost estimation study - study results. Final Technical Report, *RADC-TR-77-220*, Doty Associates, Inc., Rockville, MD (June 1977).
- [Putnam 78] Putnam, L. H. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering* 4, 4 (July 1978), 345-361.
- [Shen 84] Shen, V. Y., T. J. Yu, S. M. Thebaut and L. R. Paulsen. Identifying error-prone software: an empirical study. Submitted to *IEEE Transactions on Software Engineering* (May, 1984).
- [Thebaut 83] Thebaut, S. M. *The Saturation Effect in Large-scale Software Development: Its Impact and Control*. Ph. D. Thesis, Department of Computer Science, Purdue University (May 1983).
- [Thebaut 84] Thebaut, S. M. and Shen, V. Y. An analytic resource model for large-scale software development. *Information Processing and Management* 20, 1-2 (1984), 295-315.
- [Wang 84] Wang, A. S. *The Estimation of Software Size and Effort: An Approach Based on the Evolution of Software Metrics*. Ph. D. Thesis, Department of Computer Science, Purdue University (August 1984).

DATE  
FILMED  
— 8